Networks of Sensors

Onderwater, M.

2016

document version
Publisher's PDF, also known as Version of record

Part I

Sensor Networks

2

An Overview of Centralized Middleware Components for Sensor Networks

As described in the previous chapter, sensors are an emerging technology and will play a large role in our daily lives in the near future. In particular, the popularity of IoT is a driving factor for sensor-related applications. As discussed in Section 1.2, these applications will encounter several challenges that are typical for dealing with sensors, sensor nodes, and sensor networks. In traditional IT systems, a *middleware component* is often used to help applications deal with these challenges. The component forms a bridge between applications and sensor technology, and facilitates simpler application development.

The scientific literature contains a wide variety of such middleware components, based on ideas from, e.g., database technology, and on aspects of quality of service. Recently, much attention in literature is aimed at a special class of components that consider sensor networks that have no capacity to run part of the middleware component on the sensor nodes. In this chapter we introduce the term 'centralized' for these components, and illustrate their relevance using a literature review of existing middleware components. After a close look at non-centralized components, we describe the general architectural setup of a centralized component, and discuss four well-known examples of such a component. Finally, we identify directions for further research that will impact centralized components in the near future.

This chapter is based on the results presented in [3].

2.1 Introduction

Figure 2.1 illustrates the role of a middleware component in the context of sensors and of applications based on their data. On the left, sensors are used to, e.g., monitor the indoor climate of a house and detect smoke in an office building. The sensors report to the middleware component, and this component makes the data available to applications. The fire department can use this data to receive alarms related to the detection of smoke in the office building. Also, the owner of the house can inspect a graph of the current CO_2 levels, for which the data is obtained from the middleware component.

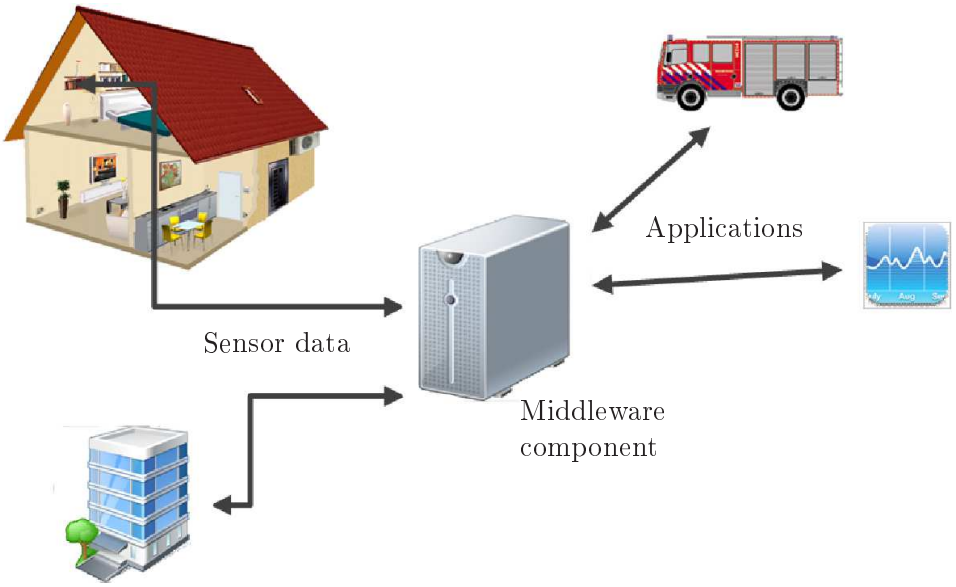

FIGURE 2.1: Illustration of how a sensor middleware component forms a natural bridge between sensors (on the left) and applications (on the right).

The middleware component forms a bridge between the sensor technology and the applications relying on sensor data. Because of this central role, the middleware component is in an ideal position to shield applications from some of the sensor-related challenges listed in Section 1.2. For instance, sensor nodes have limited resources and are often battery-powered. Excessive communication with a sensor node can quickly deplete the battery and render the node useless. A middleware component can avoid this by, e.g., providing a caching mechanism and using this to manage communication with the sensor node.

Figure 2.1 suggests that the middleware component is separated from the sensors. In this chapter we focus on components that adhere to this separation, and we introduce the term 'centralized middleware component' for them. There are, however, other approaches as well, where middleware components extend into a network of sensors and add additional intelligence to the network. These 'non-centralized' components typically require above-average computational resources, and are motivated by rapid technological developments in recent years. Despite these developments, many current sensor nodes are still simple and limited devices, and research on middleware components has slowly shifted to centralized components. Motivated by this, we review centralized middleware components for sensor networks available in the scientific literature in this chapter.

The remainder of the chapter is structured as follows. We start with a review of non-centralized middleware components in the scientific literature in Section 2.2. Then, Section 2.3 discusses the increased popularity of centralized components in recent years, and illustrates their architectural setup with a high-level outline. Following this, in Section 2.3.1, is a description of a leading web-standard for modeling sensors and, in Section 2.3.2, an overview of the most well-known centralized middleware components. Finally, we identify directions for future research in Section 2.4, and conclude the chapter in Section 2.5.

2.2 Non-centralized middleware components

Middleware components for sensor networks have received abundant attention in the scientific literature. In this section, we give an overview of the available non-centralized sensor middleware components in the literature, with an often-used categorization as a guideline. For a more in-depth review we refer to, e.g., the surveys [110, 158], or to the specialized surveys [109] (on service-oriented middleware components) and [147] (on programming sensor networks). The following categorization, adapted from [21, 60, 61, 66, 104], is often used in the literature:

- Database-inspired components,
- Virtual Machine-motivated components,
- Agent-based components,
- Application-driven components,
- Message-oriented components.

Below, we describe each category, illustrate the architectural similarities within a category, and list several middleware components contained in each category. Table 2.1 contains the middleware components per category.

Database-inspired components. This subclass of middleware components views the sensor network as a distributed database and adapts existing querying techniques to the sensor network. Figure 2.2 illustrates a simple network with three sensor nodes, and a laptop interested in collecting data from these nodes. Each node typically has a local database (DB) and a query engine for dealing with queries. Together, the storage and query processing facilities form the middleware component. As an example, suppose that the laptop issues a query to the network, requesting all measurements from nodes 2 and 3 at intervals of 1 second for the next 10 seconds. In SQL-like notation, this request is stated as *SELECT * FROM sensors WHERE id IN (2,3) SAMPLE RATE 1s FOR 10s*. The laptop passes this query to node 1, which in turn forwards it to nodes 2 and 3. There, the query is executed and the results are returned.

This example illustrates three important aspects of middleware components in this category. Firstly, issued queries should be routed to the correct nodes and thus the middleware component should maintain some structured representation of the network. Secondly, the component needs a sensor-specific query language, which usually is adapted from the SQL language used to query conventional databases. In the query above, the keywords "SAMPLE RATE" and "FOR" are examples of keywords that have been added to the traditional SQL syntax. Thirdly, in conventional database systems the results of the queries are always immediately returned after it has been processed. In the context of sensor networks, however, queries can run and produce output continuously, resulting in a stream of data. Middleware components in this category include TINYDB [98], IRISNET [55], SINA [138], COUGAR [160], DSWARE [92], SNEE [50], and KSPOT [20].

Virtual Machine-inspired components. A Virtual Machine (VM) is a platform-independent programming environment that hides details of the underlying operating system and hardware. Software developers can thus write programs in one language, and deploy it to any device running a virtual machine. Since sensor nodes are based on a wide variety of software and hardware, using a virtual machine is also appropriate for sensor networks. When each sensor node runs a virtual machine, creating reusable programs for sensor networks becomes considerably easier. Figure 2.3 illustrates the setup of a typical VM-motivated component, containing a simple sensor network of three nodes with each of them running a virtual machine. The application on the laptop sends a program into the sensor network, where it arrives at node 1. There, it

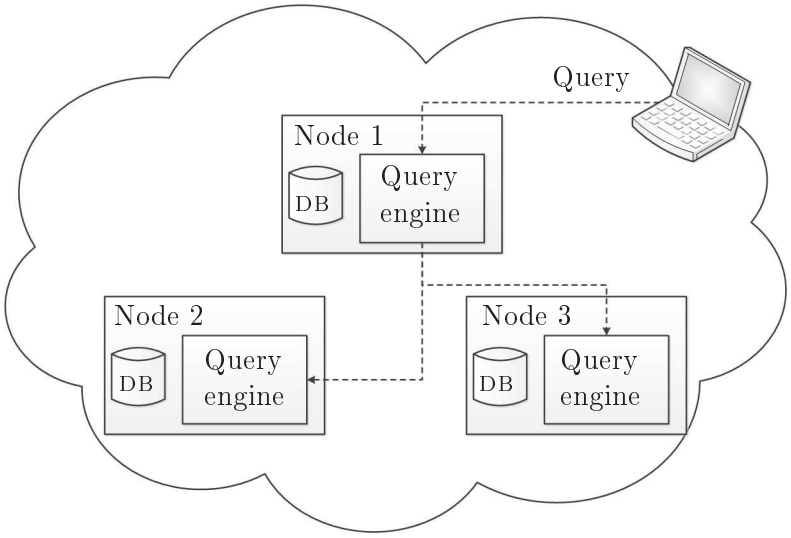

Figure 2.2: Simple sensor network where the nodes are viewed as parts of a distributed database. Each node comprises of a local database and query engine. The engine interprets and forwards queries through the network to the correct nodes, and executes them to retrieve data from its local database.

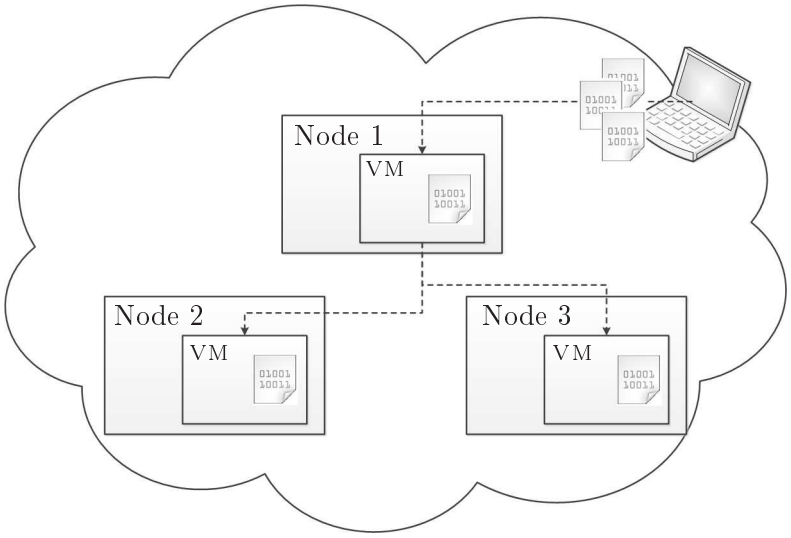

Figure 2.3: VM-based components deploy a Virtual Machine on each node, and thus provide an execution environment for small pieces of code. In this example, a program is split into three parts and forwarded to the appropriate node. There, the program is executed until it is finished.

is split into three parts and forwarded to the correct nodes. Each node then executes the part it receives, until the program ends. Applications for sensor networks typically use data from multiple sensor nodes, so a key feature of VM-motivated middleware components is that it facilitates the splitting of programs. Additionally, it sends the parts of the program to the correct sensor nodes, so the middleware component also needs to maintain a structured view of the network. Finally, the middleware component must manage software updates across the sensor network. MATÉ [90], MUSE [134], and MAGNET [94] are components in this category.

Agent-based components. Agents are small pieces of software that work together to achieve a predefined goal. Unlike conventional computer programs, agents are not activated by external commands but act autonomously based on a set of rules and on information from their environment. Additionally, agents are mobile and capable of moving from one environment to another. The principles of agent-based systems have also been applied in the context of sensor networks. To facilitate the execution and migration of agents in sensor networks, middleware components in this category typically equip each node in the network with a special execution environment (EE). This is illustrated in Figure 2.4, which represents a sensor network with three nodes and two agents, one at the second sensor node, and one en-route from node 1 to node 3. As with the VM-oriented components, agent-based middleware components need a structured view of the network, require special skills from the software developers, and must facilitate the distribution of software updates. Well-known agent-based approaches are IMPALA [95], AGILLA [49], SWAP [112], and MAPS [17].

Application-driven components. Whereas the middleware components in the previous categories were grouped by an architectural similarity, the components in this category have a common goal: optimizing Quality of Service (QoS). QoS aspects are typically application-specific, so this category is named 'application-driven'. An example of an application-driven middleware component is MILAN (Middleware Linking Applications and Networks) [65], which considers both the QoS requirements of an application and the QoS capabilities of the available sensor nodes. These requirements and capabilities are then matched to select the sensor nodes involved in the application. In MILAN, the QoS levels reflect uncertainty in sensor measurements, i.e., if faced with a choice between using a sensor node with level 0.7 and a node with level 0.8, the latter is preferred due to its higher reliability. Besides MILAN, MIDFUSION [19] and the Adaptive Middleware Component (AMC) from [72] also belong to this category.

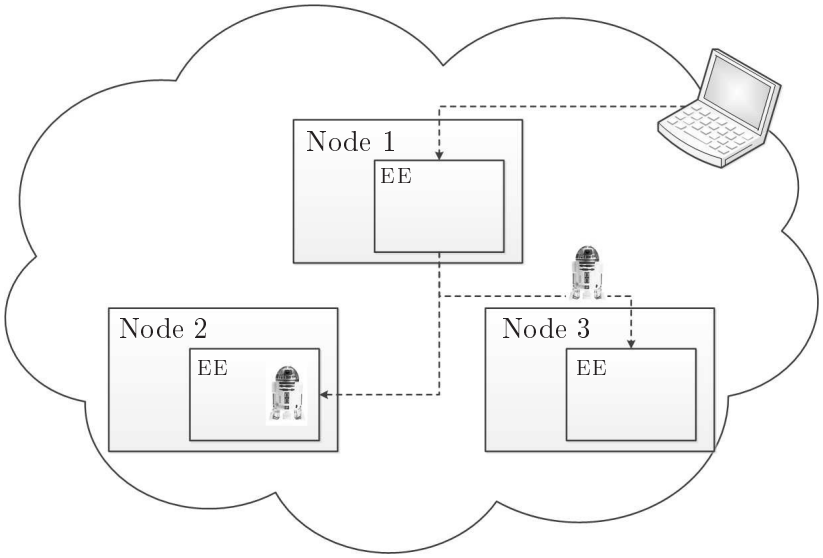

Figure 2.4: Agent-based middleware components equip each sensor node with an execution environment (EE) allowing agents to, e.g., migrate to other sensors.

Message-oriented components. In sensor networks, measurements often occur as a result of events that happen in the monitored environment. For instance, a high temperature measurement might trigger the transmission of that measurement to all interested applications. In traditional IT-systems, the *Publish/Subscribe* mechanism is often used to provide such event-driven communication, and it has also been applied in the context of sensor networks. In the Publish/Subscribe mechanism each sensor node broadcasts a standardized description of its capabilities across the network to interested applications. If an application is interested in the measurements of a sensor, it notifies the corresponding sensor node that it wants to subscribe to the sensor's events. When a sensor node detects an event, the resulting measurement is forwarded ('published') across the network to all subscribing applications. This mechanism ensures that subscribers are notified when an event occurs.

We use Mires from [145] as an example of a middleware component in this category, with Figure 2.5 illustrating the architecture of one single sensor node as used by Mires. At the bottom are the hardware components of the node, such as the sensors (measuring, e.g., temperature), the CPU, and the radio. When a new temperature measurement is done, the operating system reports it to the *Publish/Subscribe Service*. This service is at the heart of Mires,

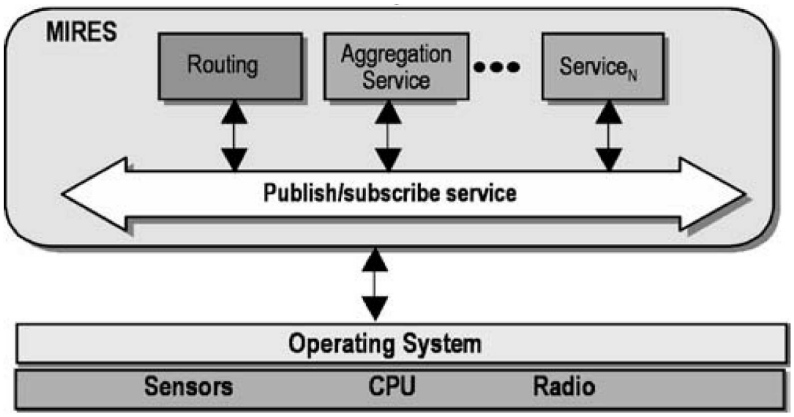

FIGURE 2.5: Architecture of one MIRES node, adapted from [145].

and provides communications between the service on the sensor node, and also to services on other nodes in the network. The measured temperature value can, e.g., be handed off to the (local) *Aggregation Service*, or it might be routed (via the *Routing Service*) to other sensor nodes. The *Publish/Subscribe* mechanism is flexible, so multiple services might be notified simultaneously of a new temperature value as well. Applications are built using MIRES by identifying what services are required from which node, and then subscribing to those services.

The event-based nature of many sensor networks makes the Publish/Subscribe mechanism a useful tool for middleware components in this category. The mechanism fully decouples applications and sensor nodes, so that adding or removing applications and/or sensor nodes does not require global changes to the middleware component. Also, the Publish/Subscribe mechanism hides vendor and hardware specific details about sensor nodes from applications. Thereby, developing applications in networks with heterogeneous sensor nodes becomes more convenient. Components AWARE [119], WMOS [162], and TINYMQ [140] also belong to this category.

2.3 Centralized middleware components

Most of the middleware components in the categories from the previous sections have not been widely adopted in practice. One of the reasons for this is that the middleware components were motivated by increased technological possibilities of sensor nodes (e.g., more memory, larger storage capacity, and

better processors), but many of the sensor nodes used in practice today are still resource-limited devices with no possibilities to run, e.g., a virtual machine. A second reason is that industry standards have received little attention in the scientific literature for sensor middleware components (e.g., Zigbee [166]). As the authors of [113] put it: *Academic WSN research and ZigBee appear to intersect only seldom, if at all. [...] This progressively caused industry to lose interest in academic* WSN *research, as compliance with standards [...] is key to industry applications.* Finally, one can question whether a sensor middleware component should extend into the sensor network. If it does, then the

Components	**Timeframe**	**Reference**	**Website**
Database-inspired			
SINA	2001	[138]	-
Cougar	2000-2005	[160]	[40]
IRISNet	2002-2005	[55]	[73]
TinyDB	2002-2005	[98]	[151]
DSWare	2004	[92]	-
KSPOT	2007-2011	[20]	[82]
SNEE	2008-2009	[50]	[144]
Virtual Machine-motivated			
Maté	2002-2005	[90]	[106]
Magnet	2001-2005	[94]	[99]
MUSE	2005	[134]	-
Agent-based			
Impala	2002-2004	[95]	-
Agilla	2004-2007	[49]	[15]
SWAP	2006	[112]	-
MAPS	2009-now	[17]	[101]
Application-driven			
MiLAN	2002-2004	[65]	-
AMC	2004	[72]	-
MidFusion	2008	[19]	-
Message-oriented			
Mires	2005	[145]	-
AWARE	2006-2009	[119]	[22]
WMOS	2011	[162]	-
TinyMQ	2011	[140]	-

Table 2.1: Overview of the middleware components per category. Each component has a reference to a paper describing the architecture, and a link to a website (if one exists).

middleware component is faced with typical network-issues such as routing, and perhaps the responsibility for this should be assigned to the network, not to the middleware component.

In recent years the scientific community has picked up on these concerns and included a new category of middleware components. Components in this category are separated from sensor technology and thus make no requirements in terms of, e.g., resources and access protocols. We call such components '*centralized middleware components*'. Figure 2.6 illustrates the role of a centralized middleware component schematically. It contains two sensor networks that are connected to a middleware component (at the bottom), and two applications that rely on sensor data (at the top). The middleware component forms a bridge between sensor technology and applications relying on sensor data, hence the term 'centralized'.

In the remainder of this section we describe several examples of centralized middleware components from the scientific literature, and discuss their advantages and disadvantages. Before that, in the next section, we describe the leading industry standard for sensor modeling, as this is included in many of the middleware components.

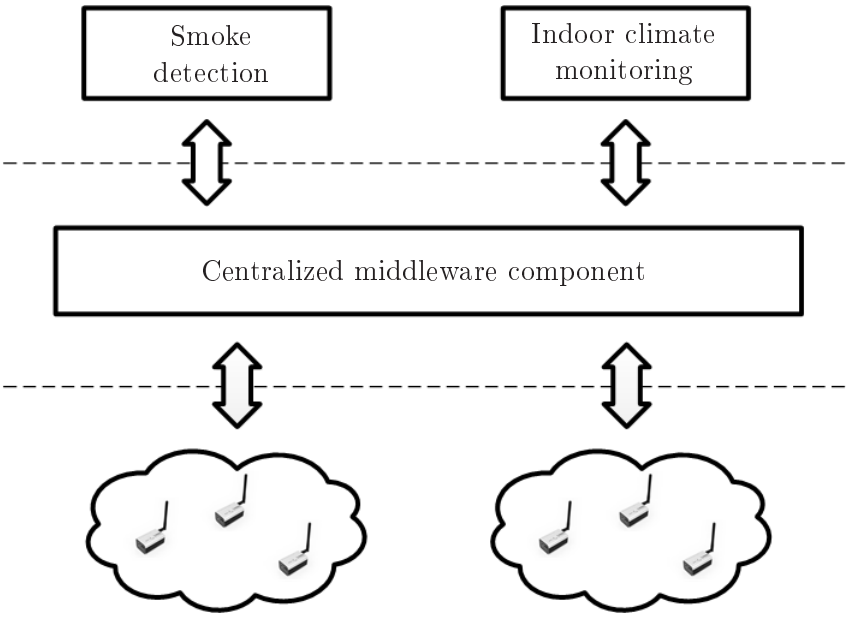

FIGURE 2.6: Position of the centralized middleware component with respect to sensor technology (at the bottom) and sensor applications (at the top).

2.3.1 Sensor Web Enablement initiative

Several years ago an international group of companies, government agencies and universities from the OpenGeospatial Consortium (OGC) [121] started the Sensor Web Enablement (SWE) initiative. This initiative aims to support the discovery and exchange of sensor information, as well as the tasking of sensor systems. It consists of standards covering the topics of modelling sensors and observations, and of interfaces for communicating with sensor nodes. The standards and interfaces in SWE are defined as web services, and include the following specifications:

- *Observations and Measurements*: a scheme for describing sensor observations and measurements.
- *Sensor Model Language*: an interface for describing sensor systems and their capabilities.
- *Sensor Observation Service* (SOS): a web service to obtain observations and sensor and platform descriptions from one or more sensor nodes.
- *Sensor Planning Service* (SPS): provides users with a standard interface for setting their own data collection requests.
- *Sensor Alert Service* (SAS): defines an interface for publishing and subscribing to sensor alerts.
- *Web Notification Service* (WNS): handles the asynchronous message delivery to the subscribers of the SAS and SPS.

Despite SWE's popularity, several drawbacks of standards have been identified in the scientific literature (from [23, 112]):

- There is no explicit ontological structure in the SWE framework.
- Security and privacy issues are not addressed.
- Conversion from a network-specific format to SWE standards requires detailed knowledge of both formats. Typically, off the shelf sensor nodes do not provide their data in SWE form. We return to this drawback in Section 2.4.
- There are no guidelines for the communication between services.
- Services are passive, so for example, a user can contact the SOS, but not vice versa.

We illustrate the SWE specifications with a use-case from the SensorSA middleware component [30], one of the components we describe in the following

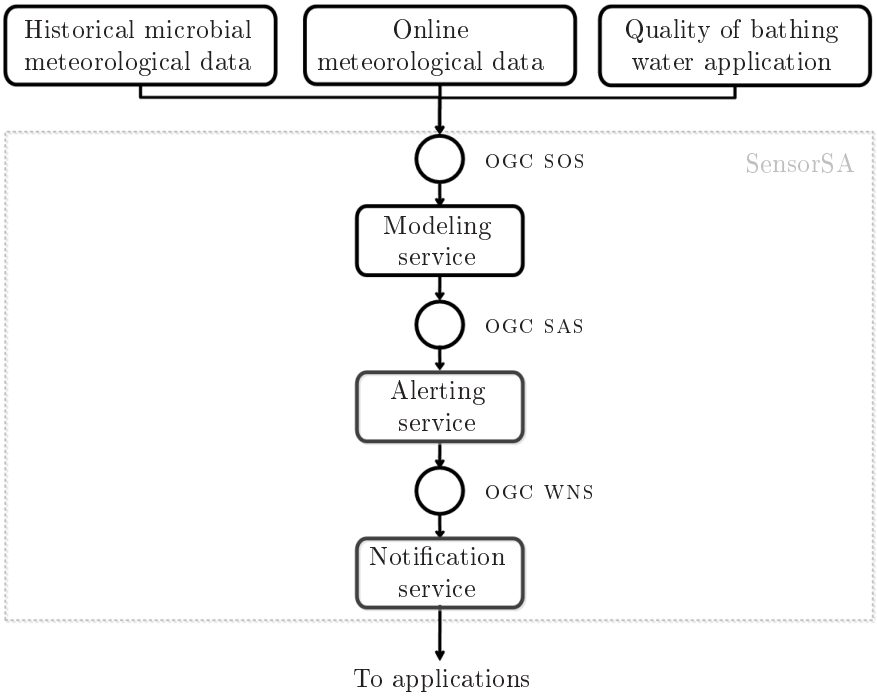

FIGURE 2.7: Abstract view of an application for monitoring the quality of seawater using the SENSORSA architecture, adapted from [30].

section. In this case study, a decision support system for marine risk management is created using SENSORSA. The system monitors the quality of seawater in areas where people often swim, and also predicts this quality for the near future. Authorities use this information to close beaches with a high risk of contamination, thereby preventing sickness due to microbial pollution.

An abstract view of the application is depicted in Figure 2.7. At the top, historical and online sensor data is imported into SENSORSA and used to update forecasting models. Importing is done using the *Sensor Observation Service*, which provides methods for requesting, filtering, and retrieving sensor data and information. The forecasting in this application is done by a (application-specific) modelling service, which generates an alarm when a prediction indicates bad water quality. This alarm is passed to the Alerting Service, an instance of the *Sensor Alert Service*. This service uses the Publish/Subscribe mechanism (which we saw earlier in Section 2.2 when discussing MIRES) to notify applications of alarms. Sending the alarm is done using *Web Notification Service*, which provides the ability to send an alarm as, e.g., an e-mail or a text message.

2.3.2 Overview of components

In this section we describe four centralized middleware components from the scientific literature. First, we describe SensorSA, because it is a good illustration of the SWE standards, and because of its clear documentation. Then we discuss SenseWeb, to demonstrate that non-SWE web services can also play a valuable role. Next, we present PULSENet, the most complete centralized middleware component available, featuring both SWE-based interfaces as well as various other industry standards for dealing with sensor data. Finally, we describe the middleware component LSM that utilizes the streaming nature of sensor data (one of the future research directions we identify in Section 2.4), and provides a way of publishing data without using web services. The four components are summarized in Table 2.2.

SensorSA

SensorSA (Sensor Service Architecture) is a middleware component developed in the SANY (Sensors Anywhere) project. SensorSA *aims to improve the interoperability of in-situ sensors and sensor networks, allowing quick and cost-efficient reuse of data and services from currently incompatible sources*[...] [30]. Its central role is illustrated in Figure 2.8, with in-situ sensors at the bottom, users at the top, and SensorSA in the middle.

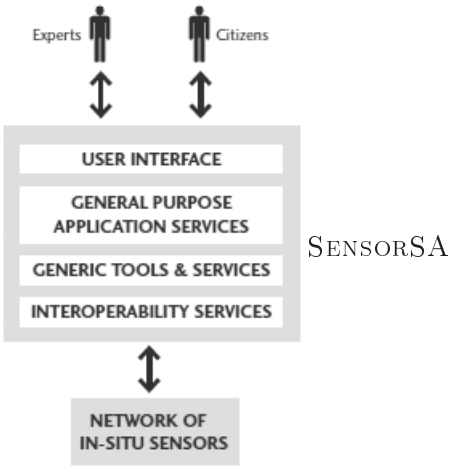

Figure 2.8: Illustration of the role of SensorSA, a centralized middleware component (adapted from [30]).

Its use of open standards from SWE makes SENSORSA an interesting component for companies seeking to include sensor data into their IT infrastructure. Several use cases are discussed in [30] and illustrate the use of SWE in practice. Additionally, SENSORSA uses several non-SWE interfaces from OGC for, e.g., visualization. SENSORSA also serves as an example of how a centralized approach simplifies security issues. Since the middleware component is not responsible for security on the sensor network (this network is considered to be owned and managed by a third party), it only needs to secure its own services. For this, SENSORSA relies on well-known security mechanisms for access control to service networks [30, Chap. 5]. SENSORSA contains several data fusion algorithms for analysis along both the time-dimension and the space-dimension of sensor data. Together with a time series toolbox for analyzing streaming sensor data, SENSORSA thus addresses two fields that we recognize as important future research directions in Section 2.4.

Despite the steps forward provided by SENSORSA, several of the drawbacks to SWE remain: there is no ontological structure, and services still seem to be passive. Moreover, there is no implementation of SENSORSA available for download, so a quick experiment with SENSORSA on an existing sensor network is not possible.

SENSEWEB

SENSEWEB is a sensor middleware component from Microsoft Research, *designed to let multiple concurrent applications share sensing resources contributed by several entities in a flexible but uniform manner* [76]. The key elements of SENSEWEB are illustrated in Figure 2.9, and strongly resemble the high-level description of a centralized middleware component from Figure 2.6. The primary building block is the *Coordinator*, which collects data via the *Sense gateway* and publishes this data to *Applications* and *Transformers*. Sensor nodes can be addressed through the Sense gateway, which provides a uniform interface for the rest of SENSEWEB, hiding any vendor-specific aspects. Transformers are components that process sensor data into other formats, for instance by calculating averages, or by creating figurative representations of data. In this way, transformers provide low-level elements that can be easily included in applications. The Coordinator consists of two separate modules, the *SenseDB* and the *Tasking Module*. SenseDB provides load-balancing facilities by analyzing requests for data to find overlap in their desired responses, and by using a cache for sensor data. Additionally, it is responsible for keeping track of the various sensors attached to the coordinator, and of their descriptions and ca-

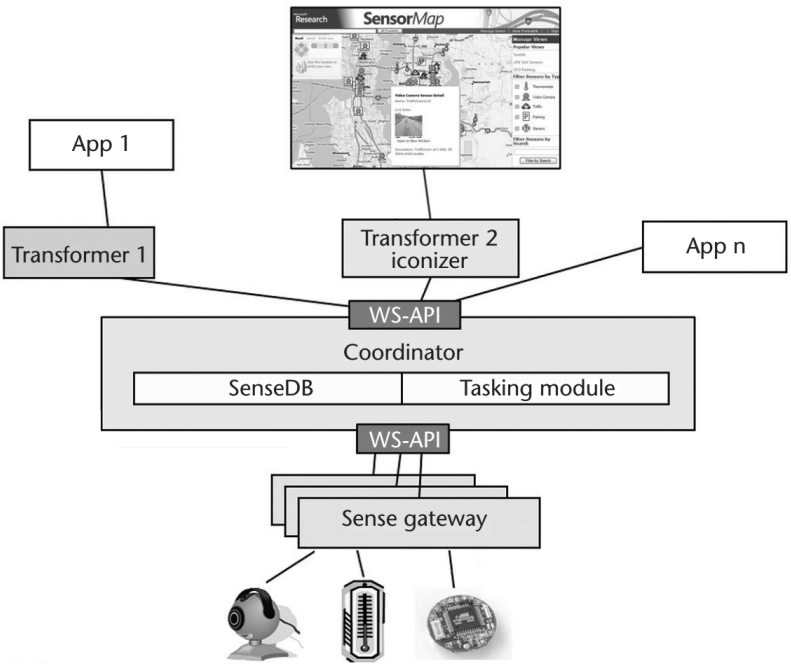

Figure 2.9: Overview of SenseWeb, illustrating how the coordinator mitigates between applications and sensor nodes (adapted from [76]).

pabilities. The Tasking Module determines which sensors are most suitable for answering a query, taking into account, e.g., bandwidth, availability, and power levels. Thus, these two models together provide an intelligent mechanism for load balancing, which is the key distinguishing feature of SenseWeb.

PULSENet

PULSENet is a sensor web component developed at the Northrop Grumman Corporation, with the objective to *provide a standards-based framework for the discovery, access, use and control of heterogeneous sensors, their metadata, and their observation data* [46]. It is based on the standards provided by SWE, supplemented by a wide variety of non-SWE standards for, e.g., describing public safety alerts, detailing military events, and visualization. Sensor networks are connected to PULSENet via plugins, which hide vendor-specific interfaces and perform the translation to and from the SWE standards. From a practical point of view, PULSENet has been tried and tested extensively. It is used, for instance, in the Defense and Intelligence domain, which contains sen-

sors and platforms with many modalities, levels of complexity, data formats, and privacy issues. Other domains include the Ocean Science community, and Air Quality applications. This emphasizes the practical relevance of PULSENet, and of centralized middleware components in general.

The authors of [46] also provide a list of best practices when dealing with SWE. These can be summarized as:

- *Apply the* SWE *standard only when needed.* Using SWE for data publishing typically means sacrificing some performance, due to SWE's complexities. So apply SWE only when a device has sufficient capacity to run web services and parse XML. Otherwise, consider using more low-level standards.
- *Keep it simple.* SWE is a large and flexible standard, offering both simple and complex data structures. Use the complex structures only if necessary to keep SWE overhead as low as possible.
- *Use [...] the* SWE *compliance tests.* OGC offers a compliance engine that allows third parties to test their implementations of SWE standards. Passing the compliance test adds significant value to a SWE-enabled middleware component.
- *Avoid reinventing the wheel.* Several open source implementations for SWE web services are available, and using them is advisable considering development time and software quality.

Unfortunately, the source code for PULSENet is unavailable from the corresponding website, so real-life experimentation with PULSENet is not possible. Also, we could not find a more detailed description of the PULSENet architecture than the one in [46].

LSM

LSM (Linked Stream Middleware) [87, 88] is a middleware component from the field of Linked Stream Data. It aims to simplify the integration of sensor data with data from other sources by providing semantic descriptions for sensor measurements and sensor data streams. As the name suggests, Linked Stream Data has two main properties: the data has mutual relationships (i.e., it is linked), and it is available via streams. The links are visible in Figure 2.10 in the Linked Data layer, and together form a complex structure of current and historical information. The data (both static and streaming) is collected in the

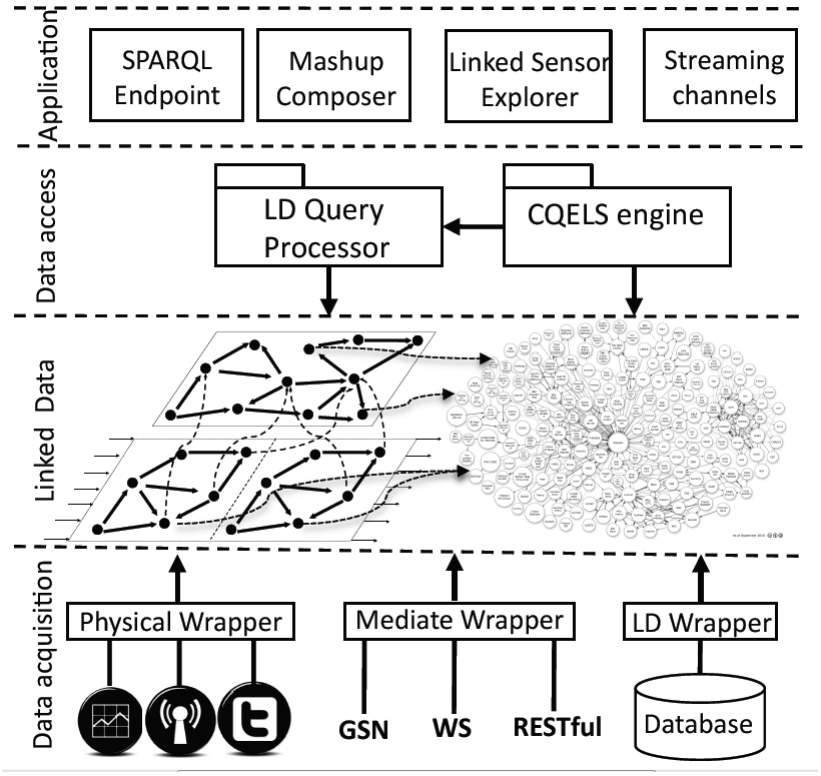

Figure 2.10: Overview of the elements of LSM, adapted from [87].

Data Acquisition layer, and transformed to a Linked Data format via 'Wrappers' (which are similar to, e.g., the plugins in PULSENet). Access to the data for applications is provided by a query processor, using a query language for streaming data: CQELS. This query language is not a standard, but developed by LSM's author in [86].

A nice feature of LSM is that it uses W3C's Semantic Sensor Network ontology [38], which also yields the relationships between data points (for instance, they can be linked via their 'location' property). By using an ontology, standard query options become available via SPARQL [128]. The query language CQELS is based on SPARQL and enables the expressiveness of an ontology for streaming data. A working demo of LSM is available online at `http://lsm.deri.ie/`. Although the concept of Linked Stream Data is promising, it is relatively new in the context of sensor networks, and therefore untested in practice. More research and experiments are necessary to demonstrate if, e.g., the CQELS query language is applicable in a broad range of applications.

Component	Timeframe	Reference	Website
SENSORSA	2006-2010	[30]	-
SENSEWEB	2006-2010	[76]	[135]
PULSENET	2009	[46]	[129]
LSM	2011-2012	[87, 88]	[96]

TABLE 2.2: Overview of four centralized middleware components described in this chapter.

2.4 Directions for future research

Centralized middleware components form a bridge between sensor technology and applications relying on sensor data. Since sensor technology is based on a wide variety of standards and protocols, a centralized middleware component should support many different technologies. The components discussed in Section 2.3 recognize this need, and provide gateways (SENSEWEB), plugins (SENSORSA, PULSENET), or wrappers (LSM) for implementing support for various technologies. Similarly, interaction with applications occurs via various different (web-)interfaces, and middleware components should support these as well. The paper describing PULSENET [46] nicely illustrates the diversity in web interfaces. From the point of view of the centralized middleware component, the large diversity of technologies can be seen as a nuisance, since it causes a lot of extra work. But support for many different technologies makes it considerably easier to develop sensor-driven applications, and is essential for the continued growth of, e.g., The Internet of Things.

A next step for centralized middleware component research concerns the 'classic' issues of Quality of Service, privacy, and security. Much research on these topics already exists (see, e.g., [36], [105], and [133, 165], respectively), but has been hampered by a lack of clear definitions in the context of sensor networks. As applications relying on sensors and sensor networks become ubiquitous, research on Quality of Service, privacy, and security will most likely regain momentum.

A third promising research direction is formed by semantic specifications of sensors. Giving a semantic description of a sensor makes it clear what type a sensor is (e.g., a 'Temperature' sensor), what units its measurements are in (e.g., 'Degrees Celsius'), and how these measurements were obtained (e.g., 'Average of 10 measurements in the last 1 second'). Such properties of sensors become particularly important once sensor data is used by third party applica-

tions, because they must understand exactly what the offered data represents. A good overview of this topic is presented in [37, 38].

Furthermore, as evidenced by LSM, techniques from the 'Data Streams' domain will become more popular. A large body of literature is available (see, e.g., [14, 51]) and is ready to be applied. Of particular interest are applications that combine streaming sensor measurements with static data from other sources, because centralized middleware components are in a unique position to collect and process both types of data.

We foresee that applications will increasingly combine sensor data with other sources of data, driven by newly available sensor data from middleware components. For instance, information from CO_2 and temperature sensor nodes in a building can be combined with data from security systems to verify alarms. If a security system gives an alarm that a burglary is in progress, an unexpected change in CO_2 and temperature measurements might confirm that something irregular is happening. This is an example of a process known as 'Data Fusion', and we think that techniques from this domain can boost the development of a new generation of innovative and intelligent sensor-related applications. Interested readers are referred to [77], which contains a review of the state-of-the-art in this domain.

2.5 Conclusion

This chapter reviewed middleware components for sensor networks in the literature using a well-known categorization. Then, we described that recently, a new category of sensor network middleware components has emerged. These components do not run part of the middleware component on the sensor nodes, contrary to many existing components. We introduced the term 'centralized middleware component' for this new type of component and discussed four examples of this type. Finally, we discussed directions for future research.